We revised syntax rules for SELECT and our comments and suggestions are as follows :

fdbs-select ::= fdbs-select-no-group | fdbs-select-group

//remove { and } since they indicate repetition[[1]](#footnote-1) in EBNF

fdbs-select-no-group ::= SELECT \* | list-of-attributes ~~| COUNT(\*)~~

FROM list-of-tables

[WHERE fdbs-where-clause]

// “e) Brackets group items together.”1 Removed COUNT(\*) because it contradicts page 5: "1. The use pf aggregate functions is limited to statements with a GROUP BY clause."

fdbs-select-group ::= SELECT table.attribute, COUNT(\*) | SUM(attribute)

FROM table GROUP BY table.attribute

list-of-attributes ::= table.attribute {,table.attribute}

list-of-tables ::= table | table1, table2

fdbs-where-clause ::= fdbs-where-join-only | fdbs-where-join-and-non-joins | fdbs-where-non-joins

fdbs-where-join-only ::= (single-join-condition)

single-join-condition ::= table1.attribute1 comparison table2.attribute2

fdbs-where-join-and-non-joins ::= (single-join-condition) AND (non-join-conditions)

fdbs-where-non-joins ::= (non-join-condition)

comparison ::= = | != | > | >= | < | <=

non-join-conditions ::= (non-join-condition) [AND | OR (non-join-condition) ]

non-join-condition ::= table1.attribute1 comparison constant

should be:

non-join-condition ::= table.attribute1 comparison table.attribute2 | constant

see page 6, paragraph 5. SELECT without Join example:

WHERE (R.A = R.C) table.attribute1 = table.attribute2

WHERE (S.D =’Kunz’**)**  table.attribute1 = constant

OR  **(**S.E = 100)

Here we added **)** and  **(** brackets, since they are defined in non-join-conditions rules

constant ::= integer-constant | string-constant

**Updated text for examples:**

page 6: „6. GROUP BY only has to be supported if there is only one table in the FROM clause. In this case there will be exactly be one grouping attribute. The only two aggregate functions you need to support~~ed is~~ are COUNT(\*) and SUM(attribute).

SELECT PERS.ANR, COUNT(\*) FROM PERS GROUP BY PERS.ANR

SELECT PERS.ANR, SUM(SALARY) FROM PERS GROUP BY PERS.ANR

**Questions for Test Queries :**

1. On page 4: „To reduce complexity horizontal partitioning will only be required on attributes of data type INTEGER“. However, in a CREPARTABS.SQL, **horizontal partitioning is applied to VARCHAR,** so how should the parser react?:

**create** **table** FLUGLINIE (

FLC **varchar**(2),

…

ALLIANZ **varchar**(20) **default** **null**,

**constraint** FLUGLINIE\_LAND\_NN **check** (LAND **is** **not** **null**),

**constraint** FLUGLINIE\_ALLIANZ\_CHK **check** (ALLIANZ **in** 'Star','Excellence','Leftover','OneWorld','SkyTeam')),

**constraint** FLUGLINIE\_FS\_HUB

**foreign** **key** (HUB) **references** FLUGHAFEN(FHC))

**HORIZONTAL (FLC('CC','KK'));**

1. You asked to not implement **SET**, in that sense what should our program do if Test Script file provided from you has this statement: **set** echo **on**; Same goes for: **alter** session **set** nls\_language = english;
2. Page 3, “Among the data types deﬁned by the SQL standard **only** **INTEGER** **and** **VARCHAR**, written in **capital letters**, have to be considered.” However, in tests scripts we see the following:
   1. **Null** is not a constant defined in grammar as |::= {integer-constant | string-constant}

**INSERT** **INTO** FLUGLINIE **VALUES** ('AB', 'D ', **null**, 'Air Berlin', **null**);

**update** FLUGLINIE **set** HUB = **NULL**;

* 1. Test Scripts have all keywords including these in small letters. How should our program respond to these statements?

1. When defining a column in Test Scripts there may be **default** **null**. These keywords are neither described in CREATE syntax nor in descriptions.
2. Since constraints with check are not to be implemented, then how to process different check constraints in scripts? On the page 4: “You have to support constraints of types PRIMARY KEY and UNIQUE”. What should a parser do with these statements:

**…**

**constraint** FLUG\_AN\_NN **check** (AN **is** **not** **null**),

**constraint** FLUG\_AB\_CHK **check** (AB **between** 0 **and** 2400),

**constraint** FLUG\_VONNACH\_CHK **check** (VON != NACH),

**constraint** FLUG\_PS **primary** **key** (FNR),

**constraint** FLUG\_FS\_FLC **foreign** **key** (FLC) **references** FLUGLINIE(FLC),

1. In this example, horizontal has attributes of String type whereas page 4 says “To reduce complexity horizontal partitioning will only be required on attributes of data type”, e.g only Integer are to be parsed requirements:

**create** table FLUGLINIE (

FLC **varchar**(2),

…

) HORIZONTAL (FLC('CC','KK'));

1. COUNT(\*) is an aggregate function used in PARDELS.SQL: **SELECT** COUNT(\*) **FROM** BUCHUNG;

However, it contradicts page 5: "1. The use pf aggregate functions is limited to statements with a GROUP BY clause."

1. This query: **SELECT** FLC, COUNT(\*) **FROM** BUCHUNG **GROUP** **BY** FLC;

from PARSELS1OR.SQL has no “BUCHUNG.” table prefix, as required by

list-of-attributes ::=table.attribute [,table.attribute]... grammar.

What must be the behavior of the parser in such case?

1. In the test script PARSELS1T.SQL there are no parentheses after WHERE:

**SELECT** \* **FROM** FLUGLINIE **WHERE** FLC = 'BA';

1. In the test script PARSELS1TGP.SQL there is a statement:

**SELECT** FLC, COUNT(\*) **FROM** FLUG **WHERE** (VON = 'FRA') **GROUP** **BY** FLC;

that has a **WHERE** clause not defined in the rule

fdbs-select-group ::= SELECT table.attribute, COUNT(\*) | SUM(attribute) FROM table GROUP BY table.attribute

1. ISO/IEC 14977 standard on EBNF http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf [↑](#footnote-ref-1)